Project: Deep RL Armm Manipulation

Reward Functions

Challenge 1:

I started the first part by implementing velocity control, and the robot reached the required accuracy of 90% around 100 iterations.

For the first challenge, the robot was given a REWARD_WIN of 20 when any part of the arm touched the object, and a REWARD_LOSS of 20 otherwise. A value of 1,10 and 20 worked almost equally well for this part. As long as the rewards were equal and opposite, the arm seemed to perform decently well, and was getting the required accuracy in around 100 to 150 iterations.

For the interim reward, the suggested average delta smoothing function was used for the reward.

avgGoalDelta = avgGoalDelta*alpha+distDelta*(1-alpha)

The alpha was set to 0.3. The results were better when a larger weight was assigned distDelta, and so the value of 0.3 was chosen.

Challenge 2:

I tried solving the challenge using velocity control, but it consistently failed for me. Switching to position control improved the performance, and after further tuning the parameters, the robot reached the required accuracy of 80% with position control.

Reaching the required accuracy in this part was much more challenging. After trying out a variety of different reward values, I reached the target by using the values REWARD_WIN = 1000 and REWARD_LOSS = -30.

The REWARD_WIN was awarded whenever the gripperbase touched the object. The REWARD_LOSS was awarded if a different part of the robot touched the object. However, if the robot used up 100 iterations, without performing any significant action, it was given a bigger negative reward of -900. These rewards seemed to work because they heavily discourage the robot from not reaching the object, and they heavily encourage the robot to touch the object with the gripperbase, without completely discouraging an incorrect contact.

The interim reward used was still the same.

avgGoalDelta = avgGoalDelta*alpha+distDelta*(1-alpha)

However I had to skew the weight much more towards the distDelta factor, so that the robot touched the object only with the gripperbase. As a result, in the end I was using an alpha value of 0.001.

Hyperparameters:

Challenge 1:

```
// Define DQN API Settings

#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN true
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.05f
#define EPS_DECAY 300

/*
/ TODO - Tune the following hyperparamete
/
*/

#define INPUT_WIDTH   128
#define INPUT_HEIGHT  128
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.2f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 16
#define USE_LSTM true
#define LSTM_SIZE 160
```

Ketan Kharche

For this challenge, I used the RMSprop optimizer, with a learning rate of 0.2. The higher learning rate seemed to work well and helped in quickly converging to the solution. I started with larger values of input width and height, as well as a larger LSTM size. However, with the bigger values, the program seemed to be running out of memory after a certain number of iterations. With the above mentioned values, I was able to run the program long enough to reach the required accuracy. I also used increased the value of EPS_DECAY to 300 to slightly prolong the randomness in actions.

Challenge 2:

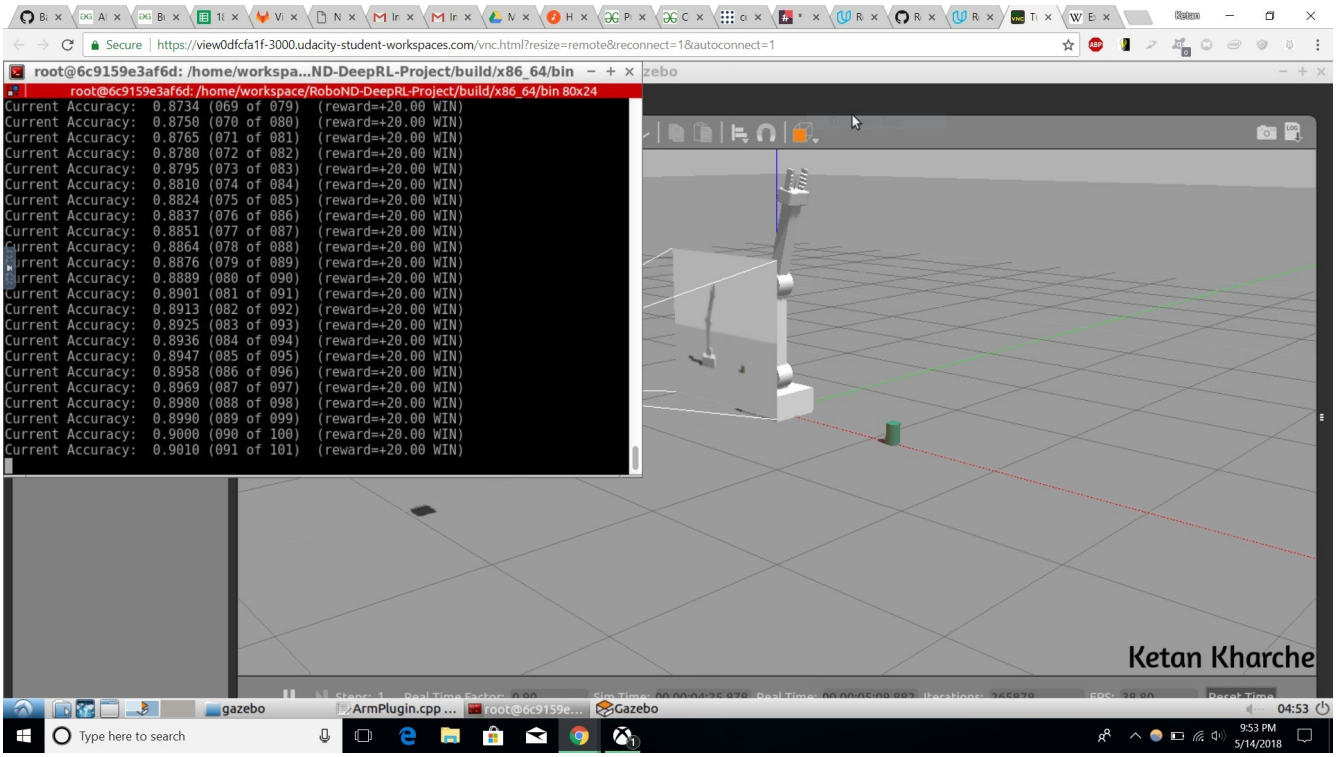
```
// Define DQN API Settings

#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN true
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.01f
#define EPS_DECAY 1000

/*
/ TODO - Tune the following hyperparameters
/
*/

#define INPUT_WIDTH   128
#define INPUT_HEIGHT  128
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 160
```

Ketan Kharche

The tuning for this part proved to be much more difficult than the first. Using the same parameters as the first part did not work too well. Eventually after a lot of tuning, the parameters shown above got me the result. I reduced the learning rate to 0.1. With the higher learning rate, the robot seemed to switch back and forth between a run of successes and failures, probably due to overcorrection of the weights. Further, I reduced the EPS_END value to 0.01, so that the robot takes even less random actions after it has learn the correct actions. The precision required for touching the object with just the gripperbase would be impacted more by the random actions. I also increased the EPS_DECAY to a much higher value of 1000. This leads to the robot trying out more random actions before converging to the solution. Thus, in a way, this extends the duration of the “learning phase” of the robot. This value was significant in increasing the accuracy for this part.

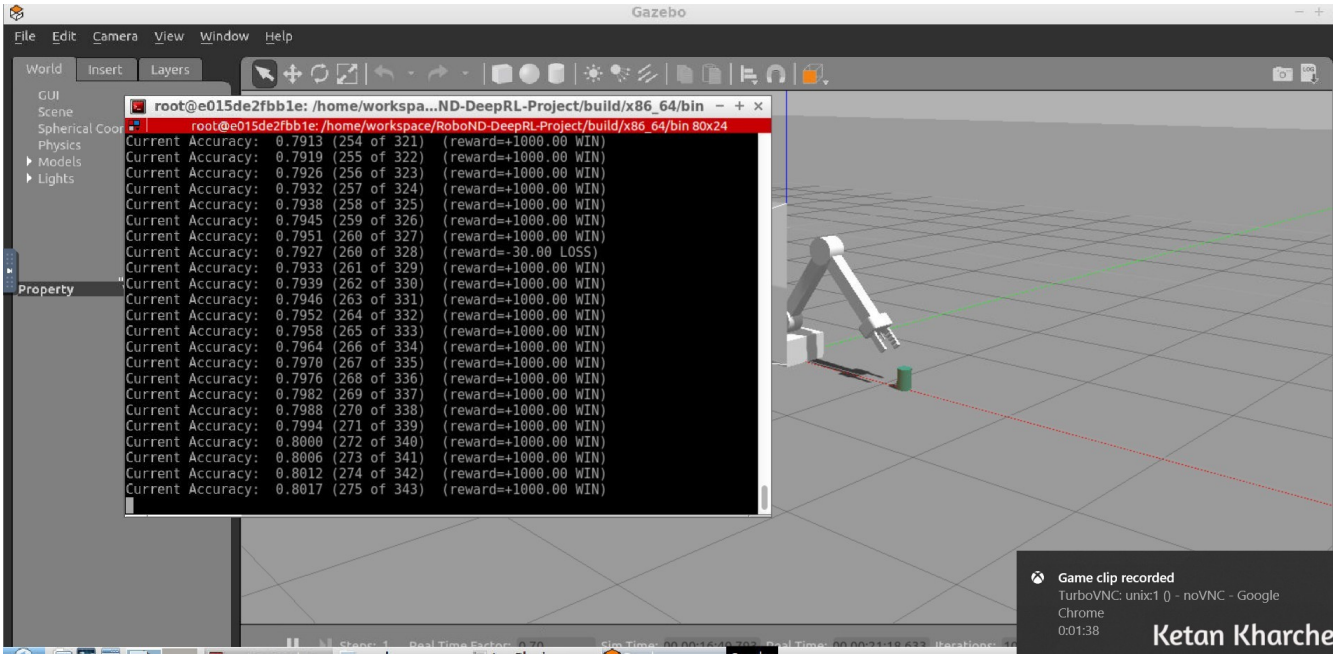
Results

Challenge 1:



For the first part, I was able to reach an accuracy of around 90% by close to a 100 iterations. The arm performed pretty well in this task, and was able to hit the target almost every time towards the end.

Challenge 2:



The performance for the second part was not as good as the first. The robot did reach the required accuracy of 80% , but took close to 350 iterations to get there. Further, even towards the end, there used to be rare rounds where it failed to reach the object. As a result, the rate of growth of the accuracy slowed down a lot towards the end. Thus, the performance of the DQN agent was not as consistent as the first round.

Future Work

The main scope of improvement is in the second challenge. The next aim is to reach the target accuracy in fewer iterations, i.e. less than 200 iterations. My next focus is modifying the interim reward function, to ensure that the gripperbase touches the object rather than any other part of the arm. Rather than using a simple smoothing function, I want to include more complex penalties for distance from the goal, the number of iterations, and for the arm staying at one place. I tried a few combinations of these, but could not get any significant improvement, I plan to explore that further.